Express Mail Label No.

UTILITY PATENT APPLICATION TRANSMITTAL
(Large Entity)

(Only for new nonprovisional applications under 37 CFR 1.53(b))

Docket No. AM9-99-0161
Total Pages in this Submission

jc759 U.S. PTO 04/17/00

TO THE ASSISTANT COMMISSIONER FOR PATENTS
Box Patent Application
Washington, D.C. 20231

Transmitted herewith for filing under 35 U.S.C. 111(a) and 37 C.F.R. 1.53(b) is a new utility patent application for an invention entitled:

SYSTEM AND METHOD FOR SCHEMA-DRIVEN COMPRESSION OF EXTENSIBLE MARK-UP LANGUAGE (XML) DOCUMENTS

JC555 U.S. PTO 09/551402 04/17/00

and invented by:

Marc Georges Girardot and Neelakantan Sundaresan

If a CONTINUATION APPLICATION, *check appropriate box and supply the requisite information:*

☐ **Continuation** ☐ **Divisional** ☐ **Continuation-in-part (CIP)** of prior application No.: _____

Which is a:

☐ **Continuation** ☐ **Divisional** ☐ **Continuation-in-part (CIP)** of prior application No.: _____

Which is a:

☐ **Continuation** ☐ **Divisional** ☐ **Continuation-in-part (CIP)** of prior application No.: _____

Enclosed are:

Application Elements

1. ☒ Filing fee as calculated and transmitted as described below
2. ☒ Specification having 41 pages and including the following:
 - a. ☒ Descriptive Title of the Invention
 - b. ☐ Cross References to Related Applications *(if applicable)*
 - c. ☐ Statement Regarding Federally-sponsored Research/Development *(if applicable)*
 - d. ☐ Reference to Microfiche Appendix *(if applicable)*
 - e. ☒ Background of the Invention
 - f. ☒ Brief Summary of the Invention
 - g. ☒ Brief Description of the Drawings *(if drawings filed)*
 - h. ☒ Detailed Description
 - i. ☒ Claim(s) as Classified Below
 - j. ☒ Abstract of the Disclosure

UTILITY PATENT APPLICATION TRANSMITTAL
(Large Entity)

(Only for new nonprovisional applications under 37 CFR 1.53(b))

Docket No. AM9-99-0161
Total Pages in this Submission

Application Elements (Continued)

3. ☒ Drawing(s) *(when necessary as prescribed by 35 USC 113)*
 a. ☐ Formal Number of Sheets _____
 b. ☒ Informal Number of Sheets **11 (Figs. 1-9)**

4. ☒ Oath or Declaration
 a. ☒ Newly executed *(original or copy)* ☐ Unexecuted
 b. ☐ Copy from a prior application (37 CFR 1.63(d)) *(for continuation/divisional application only)*
 c. ☒ With Power of Attorney ☐ Without Power of Attorney
 d. ☐ *DELETION OF INVENTOR(S)*
 Signed statement attached deleting inventor(s) named in the prior application, see 37 C.F.R. 1.63(d)(2) and 1.33(b).

5. ☐ Incorporation By Reference *(usable if Box 4b is checked)*
The entire disclosure of the prior application, from which a copy of the oath or declaration is supplied under Box 4b, is considered as being part of the disclosure of the accompanying application and is hereby incorporated by reference therein.

6. ☐ Computer Program in Microfiche *(Appendix)*

7. ☐ Nucleotide and/or Amino Acid Sequence Submission *(if applicable, all must be included)*
 a. ☐ Paper Copy
 b. ☐ Computer Readable Copy *(identical to computer copy)*
 c. ☐ Statement Verifying Identical Paper and Computer Readable Copy

Accompanying Application Parts

8. ☒ Assignment Papers *(cover sheet & document(s))*

9. ☐ 37 CFR 3.73(B) Statement *(when there is an assignee)*

10. ☐ English Translation Document *(if applicable)*

11. ☐ Information Disclosure Statement/PTO-1449 ☐ Copies of IDS Citations

12. ☐ Preliminary Amendment

13. ☒ Acknowledgment postcard

14. ☐ Certificate of Mailing
 ☐ First Class ☐ Express Mail *(Specify Label No.):* _____

DO NOT DETACH

UTILITY PATENT APPLICATION TRANSMITTAL
(Large Entity)

(Only for new nonprovisional applications under 37 CFR 1.53(b))

Docket No. AM9-99-0161
Total Pages in this Submission

Accompanying Application Parts (Continued)

15. ☐ Certified Copy of Priority Document(s) *(if foreign priority is claimed)*

16. ☐ Additional Enclosures *(please identify below):*

Fee Calculation and Transmittal

CLAIMS AS FILED

For	#Filed	#Allowed	#Extra	Rate	Fee
Total Claims	44	- 20 =	24	x $18.00	$432.00
Indep. Claims	5	- 3 =	2	x $78.00	$156.00
Multiple Dependent Claims (check if applicable) ☐					$0.00
				BASIC FEE	$690.00
OTHER FEE *(specify purpose)* Assignment Recordation					$40.00
				TOTAL FILING FEE	$1,318.00

☒ A check in the amount of **$1,318.00** to cover the filing fee is enclosed.

☒ The Commissioner is hereby authorized to charge and credit Deposit Account No. **50-0481** as described below. A duplicate copy of this sheet is enclosed.

- ☐ Charge the amount of as filing fee.
- ☒ Credit any overpayment.
- ☒ Charge any additional filing fees required under 37 C.F.R. 1.16 and 1.17.
- ☐ Charge the issue fee set in 37 C.F.R. 1.18 at the mailing of the Notice of Allowance, pursuant to 37 C.F.R. 1.311(b).

Signature

Dated: **April 17, 2000**

Sean M. McGinn, Esq.
Reg. No.: 34,386

cc:

Customer No.: 21254

MCGINN & GIBB, P.C.
A PROFESSIONAL LIMITED LIABILITY COMPANY
PATENTS, TRADEMARKS, COPYRIGHTS, AND INTELLECTUAL PROPERTY LAW
1701 CLARENDON BOULEVARD, SUITE 100
ARLINGTON, VIRGINIA 22209
TELEPHONE (703) 294-6699
FACSIMILE (703) 294-6696

DOCKET NO. AM9-99-0161

APPLICATION FOR UNITED STATES LETTERS PATENT

APPLICANT: Marc Georges Girardot and Neelakantan Sundaresan

FOR: SYSTEM AND METHOD FOR SCHEMA-DRIVEN COMPRESSION OF EXTENSIBLE MARK-UP LANGUAGE (XML) DOCUMENTS

DOCKET NO.: AM9-99-0161

SYSTEM AND METHOD FOR SCHEMA-DRIVEN COMPRESSION OF EXTENSIBLE MARK-UP LANGUAGE (XML) DOCUMENTS

BACKGROUND OF THE INVENTION

Field of the Invention

The present invention relates to an efficient compression algorithm for XML (extensible mark-up language) documents, and more particularly to a computer system, method and computer-readable code for schema-driven compression of extensible mark-up language (XML) documents.

Description of the Related Art

The Binary XML Content Format Specification (e.g., WBXML which is an acronym for wireless application protocol binary XML) defines a compact binary representation of the Extensible Markup Language (XML). ("XML" is a trademark of Massachusetts Institute of Technology.) The binary XML content format is designed to reduce the transmission size of XML documents with no loss of functionality or semantic information.

For example, it preserves the element structure of XML, allowing a browser to skip unknown elements or attributes. More specifically, it encodes the tag names and the attributes names and values with tokens (e.g., a token may be a single byte). Tokens (e.g., application

tokens) are split into a set of overlapping code spaces. A particular token's meaning is dependent on the context in which it is used. Tokens are organized in the following manner. That is, there are two classifications of tokens: global tokens and application tokens.

Global tokens are assigned a fixed set of codes in all contexts and are unambiguous in all situations. Global codes are used to encode inline data (e.g., strings, entities, opaque data, etc.) and to encode a variety of miscellaneous control functions.

Application tokens have a context-dependent meaning and are split into two overlapping code spaces. These two code spaces are the tag code space and the attribute code space. A given token value (e.g., 0x99 representing a hexadecimal value; the decimal value corresponding to the hexadecimal value 0x99 is 153) will have a different meaning depending on whether it represents a token in the tag or attribute code space. The tag code space represents specific tag names. Each tag token is a single-byte code and represents a specific tag name (e.g., CARD).

The attribute code space is split into two numeric ranges representing attribute prefixes and attribute values respectively.

Each code space (e.g., for both tag and attribute code space) is further split into a series of 256 code pages. Code pages allow for future expansion of the well-known codes. A single token (e.g., SWITCH_PAGE) switches between the code pages. The definition of tag and attribute codes is document-type-specific. Global codes are divided between a generic set of codes common to all document types and a set reserved for document-type specific extensions.

Huffman and Lempel Ziv (LZ77 and LZ78) algorithms ZLIB (Zip LIBrary) and GZIP (GNU (GNU's Not Unix) ZIP) are two implementations of these algorithms) are known for text data. However, as XML documents are compressed, the structural information is not necessarily

maintained in the compressed form so that the documents cannot be easily reconstructed. Moreover, in applying these algorithms, some (if not all) structural information cannot be retrieved without prior decompression because the compressed stream is a flat byte stream.

Further, hitherto the present invention, separating markup (e.g., structure such as element names and attribute names and values) and non-markup (data), and compressing the non-markup component using ZLIB and the markup component using binary encoding has not been performed.

Further, a binary encoding component which would retain the structure occupies approximately twice as much space as the ZLIB equivalent that loses structure. This is problematic so that there must be a tradeoff between compressing the structure component with a higher compression rate and retaining the structure.

That is, the exemplary data compression algorithms (Huffman, LZ77, LZ78, Millau (the inventive algorithm)) are lossless but traditional algorithms (Huffman, LZ) need prior decompression (a time costly operation) to retrieve the structure, whereas the inventive format does not need decompression to retrieve the structure encoded in binary format. Thus, prior to the invention, a tradeoff was required between compression rate and decompression time.

Further, the conventional methods perform poorly on relatively small documents (like eBusiness transactions) because they are designed to take advantage of the redundancy of the information which is not significant in small documents. They are not designed to take advantage of the structure. In contrast, as described below, the present invention is designed to take advantage of the structure described in the Document Type Definition (DTD) so it performs well on small documents as well as large documents.

SUMMARY OF THE INVENTION

In view of the foregoing and other problems, disadvantages, and drawbacks of the conventional methods and structures, an object of the present invention is to provide a method and system for providing an efficient compression algorithm for XML documents.

Another object of the present invention is to provide a method and system for building code spaces from information provided by a Document Type Definition (DTD) of an XML document.

In a first aspect of the present invention, a method of compressing an extensible markup language (XML) document, includes compressing an XML document such that the structural information is maintained in a compressed form to allow the document to be reconstructed. During the compression, a markup portion and a non-markup portion of the document are separated, and the non-markup component is compressed using a first compression method (e.g., preferably ZLIB or GZIP) and the markup component is compressed using a second compression method.

In a second aspect of the present invention, a method of compressing an XML document, includes creating the data structure on the server and on the client, parsing the document type definition (DTD) of the XML document, filling in the server code spaces, and filling in the client code spaces.

Hence, the present invention not only takes advantage of the separation of the structure and data of an XML document, but it also takes advantage of the associated DTD (schema) of the document to perform optimization. The DTD schema describes the constraints on the structures,

possible values, and occurrence restrictions of attribute values and elements. The compression algorithm, once it knows that there is a schema associated with the document, takes advantage of this and produces further compression of the data.

Further, the present invention provides an efficient compression algorithm for XML documents in which the XML documents are compressed, and such that the structural information will be kept in the compressed form so that the documents can be easily reconstructed. The invention provides a lossless compression algorithm that gets as close as possible to the ZLIB algorithm. Hence, with the invention, the markup (structure) and non-markup (data) can be separated , and the non-markup component can be compressed using ZLIB and the markup component can be compressed using binary encoding.

Thus, with the unique and unobvious aspects of the present invention, the schema information (the DTD associated with the document) can be used to compress the structure component and obtain higher compression rate while simultaneously retaining the structure.

Further, the method and system of the present invention provide better compression rates for small documents (like eBusiness transactions) than GZIP and other conventional schemes.

BRIEF DESCRIPTION OF THE DRAWINGS

The foregoing and other purposes, aspects and advantages will be better understood from the following detailed description of a preferred embodiment of the invention with reference to the drawings, in which:

Figure 1 illustrates a flowchart for the method according to the present invention;

Figure 2 illustrates a schematic block diagram of a system for implementing the present invention;

Figure 3A illustrates a flowchart for a method 300 of compression according to the present invention;

Figure 3B illustrates an exemplary hash table for use with the invention;

Figure 3C illustrates an exemplary code space array for use with the invention;

Figure 4 illustrates a flowchart of the operation 400 of step 330 of the compression method 300 according to the present invention;

Figure 5 illustrates a method 500 of filling in a two-dimensional array for an element code space according to the present invention;

Figure 6 illustrates a method 600 of filling in an unproved code space according to the invention;

Figure 7 illustrates an exemplary information handling/computer workstation 700 for use with the invention;

Figure 8 illustrates an exemplary data processing network for use with the invention; and

Figure 9 illustrates a medium 800 for storing a program for implementing the method according to the present invention.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS OF THE INVENTION

Referring now to the drawings, and more particularly to Figures 1- 9, there are shown preferred embodiments of the method and structures according to the present invention.

Generally, the invention provides an efficient compression algorithm for XML documents.

As mentioned above, ZLIB algorithms are known for text data. As XML documents are compressed, the structural information will be kept in the compressed form so that the documents can be easily reconstructed. However, such compression and reconstruction is inefficient. Given that the lower bound on the compression size is what can be obtained by applying ZLIB where ZLIB loses all structural information, the invention provides a lossless compression algorithm that gets as close as possible to ZLIB.

With the invention, it is possible to separate markup (structure) and non-markup (data), and compress the non-markup component using ZLIB and the markup component using binary encoding. It is noted that binary encoding is described in the WAP Binary XML Content Format Specification, W3C Note, 24 June 1999. The present invention has added enhancements to be able to separate content from structure and link them together. More specifically, the invention provides information in the structure stream to retrieve data from the content stream.

Further, the binary encoding component which retains the structure occupies approximately twice as much space (e.g., in the worst case but it can be smaller for small documents) as the ZLIB equivalent that loses structure. However, this is not truly a drawback

because the structure stream is usually small compared to the data stream. Moreover, it can be compressed using a traditional compression algorithm, thus achieving better compression rate than with the traditional algorithm alone. In the invention, the schema information (the DTD associated with the document) can be used to compress the structure component and obtain higher compression rate while simultaneously retaining the structure.

An XML document primarily is formed of a strictly nested hierarchy of elements with a single root. In the context of the present application, "strictly nested" means that an XML document can be represented as a tree. Elements can contain character data, child elements, or a mixture of both. In addition, the elements may have attributes (e.g., characteristics or properties). Child character data and child elements are strictly ordered (e.g., if element A is defined in the DTD with children B, C and D in this order, then chidden B, C and D must appear in this order in the XML document) whereas attributes are not. For example, consider:

```
<?xml version = "1.0"?>
<Book Author = "John Steinbeck" Genre = "literature">
 <Title> Of Mice and Men</Title>
 <Chapter id = "1">
    Blah...Blah...
 <Chapter id = "2">
    Blah...Blah...
 </Chapter>
</Book>
```

The names of the elements and attributes and their order in the hierarchy (among other things) form the XML markup language used by the document. This language can be defined by the document author or it can be inferred from the document's structure. In the example shown above, the language contains three elements: Book, Title, and Chapter. The Book element contains a single Title element and one or more Chapter elements. The Book element has an Author attribute and a Genre attribute and the Chapter element has an id attribute (e.g., "2").

An important reason to explicitly define the language is so that documents can be checked (e.g., both before and after compression because the inventive format retains the structure) to determine whether the documents conform to the language (e.g., such as XML). For example, if a grammar for the Book language was defined, authors using this grammar could use a validating parser to ensure that their documents conformed to the language.

An XML markup language is defined in a Document Type Definition (DTD). The DTD is either contained in a <!DOCTYPE> tag, contained in an external file and referenced from a <!DOCTYPE> tag, or both. For example, the document shown above could contain the following <!DOCTYPE> tag:

```
<!DOCTYPE Book [
 <!ELEMENT Book (Title, Chapter+)>
 <!ATTLIST Book Author CDATA #REQUIRED
                Genre (literature|science|history) #REQUIRED>
 <!ELEMENT Title (#PCDATA)>
 <!ELEMENT Chapter (#PCDATA)>
```

<!ATTLIST Chapter id ID #REQUIRED>

]>

An element is defined as a group of one or more subelements/subgroups, character data, EMPTY, or ANY. For example:

Group:

<!ELEMENT A (B,C) >

Character data:

<!ELEMENT A (#PCDATA) >

EMPTY:

<!ELEMENT A EMPTY>

ANY:

<!ELEMENT A ANY>

Elements can have zero or more attributes. For example:

<!ELEMENT A (#PCDATA)>

<!--Declare an attribute a for element A-->

<!ATTLIST A a CDATA #IMPLIED>

Attributes can be optional, required, or have a fixed value. Optional attributes may have a default, but fixed attributes must have a default. For example:

Optional without a default:

<!--Element A has an attribute a. #IMPLIED = "optional, no default"-->

<!ATTLIST A a CDATA #IMPLIED>

Optional with a default:

<!If attribute a is not provided, a default of "aaa" will be used.-->

<!ATTLIST A a CDATA "aaa">

Required:

<!ATTLIST A a CDATA #REQUIRED>

Fixed:

<!-- The value of attribute a is always "aaa"-->

<!ATTLIST A a CDATA #FIXED "aaa">

Each attribute has a type:

Character data:

<!ATTLIST A a CDATA #IMPLIED>

A user-defined enumerated type

<!--Attribute a uses a simple enumeration.-->

<!ATTLIST A a (yes|no) #IMPLIED>

<!--Attribute a uses an enumeration of notation types.

See the XML specification for complete details. -->

<!ATTLIST A a NOTATION (ps | pdf) #IMPLIED>

ID, IDREF: These attributes point from one element to another. The value of the IDREF attribute on the pointing element is the same as the value of the ID attribute on the pointed-to element. For example, consider:

<!--Attribute id gives the ID of element A-->

<!ATTLIST A id ID #IMPLIED>

<!--Attribute ref points to the ID of another element-->

<!ATTLIST A ref IDREF #IMPLIED>

ENTITY, ENTITIES: These attributes point to external data in the form of unparsed entities. For complete details, see the XML specification (e.g., see Extensible Markup Language (XML) 1.0 Specification, W3C Recommendation, 10 February 1998).

<!--Attribute a points to a single unparsed entity-->

<!ATTLIST A a ENTITY #IMPLIED>

<!--Attribute b points to multiple unparsed entities-->

<!ATTLIST A b ENTITIES #IMPLIED>

NMTOKEN, NMTOKENS. These attributes have single/multiple tokens as values.

<!ATTLIST A a NMTOKEN #IMPLIED>

<!ATTLIST A b NMTOKENS #IMPLIED>

The DTD specifies constraints on the structure and the type of the XML documents. Hence, given a DTD, it is possible to know that the document will follow certain constraints on structure and types. This can be used to further compact (e.g., compress) the representation of the XML structure in the binary form (e.g., binary encoding).

The Method of the Invention

Compression

Referring now to Figure 1, first a flow diagram of a method 100 according to the invention is provided which shows how code spaces are used to compress an XML structure.

First, in step 110, the server gets the DTD of the document requested by the client. It builds the code spaces from this DTD according to a specific data structure.

Then, in step 120, the server parses the document. Each time it encounters an element or attribute name or value, it searches its corresponding token in the code spaces and transmits it to the client.

In step 130, the client gets the DTD of the document it requested. It builds the code spaces from this DTD according to a specific data structure (e.g., different from the server).

In step 140, the client receives the document. Each time it encounters an element or attribute name or value token, it searches its corresponding text string in the code spaces in order to decode the binary structure.

Figure 2 illustrates a functional system block diagram for implementing the invention.

More specifically, the input can be either an XML stream 210 or a Document Object Model (DOM®) tree 220. If it is an XML stream, this stream is spliced into two substreams, a structure stream 230 and a data stream 240. Similarly, if the input is a DOM® tree ("DOM" is a registered trademark of Massachusetts Institute of Technology), the system can produce two streams from this DOM® tree (e.g., a structure stream 230 and a data stream 240).

"DOM" is an acronym for "Document Object Model", which is a language-independent application programming interface ("API") for use with documents specified in markup languages including XML. DOM is published as a Recommendation of the World Wide Web Consortium, titled "Document Object Model (DOM) Level 1 Specification, Version 1.0" (1998) and available on the World Wide Web at http://www.w3.org/TR/REC-DOM-Level-1.

"DOM tree" refers to the logical structure with which a document is modeled using the DOM. A DOM tree is a hierarchical representation of the document structure and contents. Each DOM tree has a root node and one or more leaf nodes, with zero or more intermediate nodes, using the terminology for tree structures that is commonly known in the computer programming art. A node's predecessor node in the tree is called a "parent" and nodes below a given node in the tree are called "child" nodes.

The DOM API enables application programs to access this tree-oriented abstraction of a document, and to manipulate document structure and contents (that is, by changing, deleting, and/or adding elements). Further, the DOM enables navigating the structure of the document.

To reduce the size of the structure stream without losing any information, the system encodes the structure in WBXML format. This process generates a WBXML stream 250 that is smaller (about 80% smaller) than the initial structure stream 230. Similarly, the data stream 240 is compressed using a conventional compression algorithm (e.g., GZIP). This produces a smaller stream called compressed data stream 260. In an efficient embodiment of the present invention, the two first steps can be realized at the same time (e.g., performed in parallel).

The decoding can produce two outputs according to the needs of the application. That is, the system can generate a SAX event 270 for each received element. Possibly, it can create an

XML stream from these SAX events. The system can also build dynamically a DOM tree 280. Each time an element is received, it is added dynamically to the DOM tree.

The decompression of the data is done using the conventional decompression algorithm corresponding to the algorithm used for the compression. It produces a data stream 290. Then, this data can be incorporated in the DOM tree or can be used to generate SAX characters events. In an efficient embodiment of the present invention, these steps can be realized at the same time (e.g., in parallel).

Turning now to the flow diagram in Figure 3A, the primary steps of a method 300 of the present invention will be described. These steps include creating the data structure on the server and on the client, parsing the DTD, filling in the server code spaces, and filling in the client code spaces.

In step 310, the data structure on the server is created. In step 340, the data structure on the client is created. That is, the data structure on the server and on the client are described respectively.

On the server side, for a specific tag name or attribute name or value (e.g., which are of string type), the server must find the corresponding token. Thus, the server must be able to quickly find a string in a table. Hence, in an exemplary embodiment, a Hash table may be used (e.g., see Figure 3B for an exemplary hash table) where the keys are the strings and the value are the corresponding tokens.

On the client side, the client must find the tag name or attribute name or value corresponding to a specific token. More specifically, given a page number and an index in a code space page, it must find the corresponding string. The preferred data structure here is for each

code space to have a two-dimensional array indexed by page numbers and indexes in pages (e.g., see Figure 3C).

Then, in step 320, the DTD is parsed on the server and, in step 350, on the client. Preferably, the DTD is parsed using IBM XML Parser for JAVA® 1.1.16 which generates a tree structure for the DTD. JAVA® is a trademark of Sun Microsystems, Inc. (Parsing a DTD with IBM XML Parser for Java produces a tree where elements and attributes can be accessed by name.) It allows accessing elements declarations and attributes declarations with name and type and also attributes values when attribute type is enumerated. For example, method getElementDeclarations returns an enumeration of all element declarations in this DTD, getContentModel(elementName) returns the content model for the specified element name in this DTD, getAttributeDeclarations(elementName) returns an enumeration of all attribute list declarations for the specified element name in this DTD.

In step 330, the server code spaces are filled-in. This step is described in further detail below with regard to Figure 4.

For example, filling in the server code spaces is performed by the server filling in the hash table for an element code space. That is, in step 410, the page number variable is set to 0 and the index variable to 5 (e.g., in the exemplary implementation the first four indexes are reserved for global tokens).

For each element declaration (step 415), the server system gets the element name, adds it in the hash table with the element name as the key and (256 x pageNumber + index) as the value (step 420). Then, the system increments the index by 1 (step 425). The size of a page for elements is 64 because the last two bits of the index are reserved so that when the index reaches

the value 64 ("YES" in step 430), the system increments the page number by 1 and resets the index to 5 (step 435). When the page number reaches its maximum value 255, an exception is raised such that an error message is displayed.

For the attribute code space, for each element declared, the server system obtains the corresponding attribute declaration from the previously built DOM tree (step 440). It adds the attribute name in the hash table with the attribute name as the key and (256 x pageNumber + index) as the value (step 445).

If the attribute type is enumerated (e.g., enumerated attribute types are NOTATION or NAME_TOKEN_GROUP) (step 465), then the system looks for the values of this enumerated attribute (step 470). For each value, it adds the attribute value in the hash table with the attribute value as the key and (256 x pageNumber + index) as the value (step 475). Then, the system increments the index for the value by 1 (step 480). The size of a page for attribute value is 128 so that when the index reaches the value 128 (step 485), the system increments the page number by 1 and resets the index to 5 (step 490).

When the page number reaches its maximum value 255, an exception is raised. If there are no values or when the values have been successfully added to the attribute value code space, the system increments the index for the name by 1 (step 450). The size of a page for an attribute name is 128 so that when the index reaches the value 128 ("YES" in step 455), the system increments the page number by 1 and resets the index to 5 (step 460). When the page number reaches its maximum value 255, an exception is raised and an error message is displayed.

Finally, returning to Figure 3A, in step 360, the client code spaces are filled-in. A method 500 for filling in the 2- dimensional array for element code space will be described hereinbelow.

First, in step 510, the page number variable is set to 0 and the index variable to 5 (e.g., the first four indexes are reserved for global tokens). For each element declaration found (step 515), the system obtains the element name, and adds it in the elements array at a predetermined position (e.g., page number, index) (step 520). Then, the system increments the index by 1 (step 525). The size of a page for elements is 64 because the last two bits of the index are reserved so that when the index reaches the value 64 ("YES" in step 530), the system increments the page number by 1 and resets the index to 5 (step 535). When the page number reaches its maximum value 255, an exception is raised.

For an attribute code space, for each element is declared, the client's system obtains the corresponding attribute declaration from the previously built DOM structure (step 540). Then, the system adds the attribute name in the attribute names array at a predetermined position (page number, index) (step 545). If the attribute type is enumerated (e.g., enumerated attribute types are NOTATION or NAME_TOKEN_GROUP) (step 565), then the system checks for the values of this enumerated attribute (step 570).

For each value, it adds the attribute value in the attribute values array at a predetermined position (page number, index) (step 575). This system increments the index for the value by 1 (step 580). The size of a page for an attribute value is 128. Thus, when the index reaches the value 128, the system increments the page number by 1 and resets the index to 5 (step 590). When the page number reaches its maximum value 255, an exception is raised. If there are no values or when the values have been successfully added to the attribute value code space, the system increments the index for the name by 1 (step 550). The size of a page for attribute name is 128 so that when the index reaches the value 128, the system increments the page number by 1

and resets the index to 5 (step 560). When the page number reaches its maximum value 255, an exception is raised.

Improvements to Code Space

Improvement for the attribute code spaces can be provided by merging the attribute names code space and the attribute values code space into one token. That is, each couple (e.g., attribute name, attribute value) is made into a single token instead of two tokens (e.g., name and value). Thus, if an attribute has 3 possible values, then there will be three different tokens for this attribute.

Referring now to Figure 6, to fill this improved code space, for each element declared, the inventive system gets the corresponding attribute declaration from the previously-built DOM structure. If the attribute-type is not enumerated (e.g., no specific value is declared for this attribute) (a "NO" in step 645), then the system adds the attribute name in the attribute code space (e.g., hash table for the server, array for the client) (step 650).

If the attribute-type is enumerated (a "YES" in step 645), then the system looks for the values of this enumerated attribute (step 670). For each value, it adds the couple (attribute name, attribute value) with a specific token in the attribute code space (step 675).

Thereafter, when the server comes across an attribute with a value, it will look in the attribute code space for the couple (attribute name, attribute value). If the server can find the couple, the server will send this token associated therewith.

If the server cannot find it, the server will look for the attribute name in the attribute code space. If the name is found, the server sends the corresponding token for this name followed by a

string inline token followed by the attribute value encoded in the charset specified at the beginning of the inventive stream. If the name is not found, an exception is raised such that an error message is displayed and the process terminates.

Regarding improving the element code space, as mentioned above, some elements can have required or fixed attributes. For these elements, it is unnecessary to transmit tokens for the required or fixed attributes.

In order not to transmit tokens for required or fixed attributes in order to save bandwidth, the inventive system can store in the element code space the names of the required or fixed attributes with the element name. For example, attributes Author and Genre are required for element Book so the element code space stores the triplet (Book, Author, Genre) at the entry Book.

Hereinbelow is described the method used by the inventive system to fill in this element code space. That is, for each element declaration, the system gets the element name from the array on the client side, and the hash table on the server side, and the required and fixed attributes from the array on the client side, and the hash table on the server side. It adds the element names and the required and fixed attribute names to the element code space. For the fixed attributes, it also adds their value. Thus, the system knows which attributes are fixed. In the attribute code space, only the implied attributes will be stored with their value if defined as described above.

Hereinbelow is provided some exemplary code of the present invention.

```
/**
*   Builds Tag Code Space and Attribute Code Space from a <code>DTD</code> object.
*
```

```
 * @param dtd a <code>DTD</code> object
 */
private void build(DTD dtd)
  throws CodespaceException {
  System.out.println("Processing OTO
  // Read elements names
  Enumeration elementDecl = dtd.getElementDeclarations();
  int pageTag = 0, indexTag = 5, pageAtt = 0, indexAtt = 5;
  while (elementDecl.hasMoreElements()) {

      String tagName = ((ElementDecl)(elementDecl.nextElement())).getName();
      tagNames.put(tagName, new Integer(256*pageTag+indexTag));
      if ((++indexTag)= =TAG_PAGE_SIZE) {
          if ((++pageTag)= =SPACE_SIZE) throw new CodeSpaceException("Too many
tags");
          indexTag = 5;
      }

      // Read attributes names and values
      Enumeration attDefs = dtd.getAttributeDeclarations(tagName);
      while (attDefs.hasMoreElements()) {
          AttDef attDef = (AttDef)(attDefs.nextElement());
```

```
String attName = attDef.getName();
if ((attDef.getDeclaredType()= =AtDef.NAME_TOKEN_GROUP)II
    (attDef.getDeclaredType()= =AttDef.NOTATION)) {
        Enumeration enum = attDef.elements():
        if (enum= =null) {
            if (!attNames.containsKey(attName)) {// attribute name not yet in
table
            AttNames.put(attName, new Integer(256*page Att+indexAtt));

//put it
            if ((++indexAtt)= =ATT_PAGE_SlZE) {
                if ((++pageAtt)=
=SPACE_SIZE) throw new
CodeSpaceException("Too many attributes");
                indexAtt = 5;
            }
        }
    } else {
        while (enum.hasMoreElements()) {
String attstart = attName + ."=." + enum.nextElement():
if (!attNames.containsKey(attStart)) {attNames.put(attstart, new
if ((++indesAtt)= =ATT PAGE_SIZE) {
```

```
if ((++pageAtt)= =SPACE_SIZE) throw new CodespaceException("Too many attributes");
indexATT = 5;
                    }
                }
            }
        }
}else{
if (!attNames.containsKey(attName)) { // attribute name not yet in table
attNames.put(attName, new Integer(256*pagMt+indeXAn)); // put it
if ((++indexAtt)= =ATT_PAGE_SIZE) {
if ((++pageAtt)= =SPACE_SIZE) throw new CodeSpaceException("Too many attributes");
                    indexAtt = 5;
                }
            }
        }
    }
}
```

Figure 7 illustrates a representative workstation hardware environment in which the present invention may be practiced. The environment of Fig. 7 comprises a representative single

user computer workstation 700, such as a personal computer, including related peripheral devices. The workstation 700 includes a microprocessor 720 and a bus 740 employed to connect and enable communication between the microprocessor 720 and the components of the workstation 700 in accordance with known techniques. The workstation 700 typically includes a user interface adapter 760, which connects the microprocessor 720 via the bus 740 to one or more interface devices, such as a keyboard 780, mouse 721, and/or other interface devices 722, which can be any user interface device, such as a touch sensitive screen, digitized entry pad, etc. The bus 740 also connects a display device 741, such as an LCD screen or monitor, to the microprocessor 720 via a display adapter 761. The bus 740 also connects the microprocessor 720 to memory 781 and long-term storage 702 which can include a hard drive, diskette drive, tape drive, etc.

The workstation 700 may communicate with other computers or networks of computers, for example via a communications channel or modem 723. Alternatively, the workstation 700 may communicate using a wireless interface at 724, such as a CDPD (cellular digital packet data) card. The workstation 700 may be associated with such other computers in a local area network (LAN) or a wide area network (WAN), or the workstation 700 can be a client in a client/server arrangement with another computer, etc. All of these configurations, as well as the appropriate communications hardware and software, are known in the art.

Figure 8 illustrates a data processing network 800 in which the present invention may be practiced. The data processing network 800 may include a plurality of individual networks, such as wireless network 820 and network 840, each of which may include a plurality of individual workstations 700. Additionally, as those skilled in the art will appreciate, one or more LANs may

be included (not shown), where a LAN may comprise a plurality 6f intelligent workstations coupled to a host processor.

Still referring to Figure 8, the networks 820 and 840 may also include mainframe computers or servers, such as a gateway computer 846 or application server 847 (which may access a data repository 848). A gateway computer 846 serves as a point of entry into each network 840. The gateway 846 may be preferably coupled to another network 820 by means of a communications link 850a. The gateway 846 may also be directly coupled to one or more workstations 700 using a communications link 850b, 850c. The gateway computer 846 may be implemented utilizing an Enterprise Systems Architecture/380 available from the International Business Machines Corporation ("IBM"), an Enterprise Systems Architecture/390 computer, etc. Depending on the application, a midrange computer, such as an Application System/400 (also known as an AS/400) may be employed. ("Enterprise Systems Architecture/370" is a trademark of IBM; "Enterprise Systems Architecture/390", "Application System/400", and "AS/400" are registered trademarks of IBM.)

The gateway computer 846 may also be coupled 849 to a storage device (such as data Repository 848). Further, the gateway 846 may be directly or indirectly coupled to one or more workstations 700.

Those skilled in the art will appreciate that the gateway computer 846 may be located a great geographic distance from the network 820, and similarly, the workstations 700 may be located a substantial distance from the networks 820 and 840. For example, the network 820 may be located in California, while the gateway 846 may be located in Texas, and one or more of the workstations 700 may be located in New York. The workstations 700 may connect to the

wireless network 820 using a networking protocol such as the Transmission Control Protocol/internet Protocol ("TCP/IP") over a number of alternative connection media, such as cellular phone, radio frequency networks, satellite networks, etc. The wireless network 820 preferably connects to the gateway 846 using a network connection 850a such as TCP or UDP (User Datagram Protocol) over IP, X.25, Frame Relay, ISDN (Integrated Services Digital Network), PSTN (Public Switched Telephone Network), etc. The workstations 700 may alternatively connect directly to the gateway 846 using dial connections 850b or 850c. Further, the wireless network 802 and network 840 may connect to one or more other networks (not shown), in an analogous manner to that depicted in Figure 8.

Software programming code which embodies the present invention is typically accessed by the microprocessor 720 of the workstation 700 and server 847 from long-term storage media 702 of some type, such as a CD-ROM drive or hard drive. The software programming code may be embodied on any of a variety of known media for use with a data processing system, such as a diskette, hard drive, or CD-ROM. The code may be distributed on such media, or may be distributed to users from the memory or storage of one computer system over a network of some type to other computer systems for use by users of such other systems. Alternatively, the programming code may be embodied in the memory 848, and accessed by the microprocessor 720 using the bus 740. The techniques and methods for embodying software programming code in memory, on physical media, and/or distributing software code via networks are well known and will not be further discussed herein.

A user of the present invention may connect his computer to a server using a wireline connection, or a wireless connection. Wireline connections are those that use physical media

such as cables and telephone lines, whereas wireless connections use media such as satellite links, radio frequency waves, and infrared waves. Many connection techniques can be used with these various media, such as: using the computer's modem to establish a connection over a telephone line; using a LAN card such as Token Ring or Ethernet; using a cellular modem to establish a wireless connection; etc. The user's computer may be any type of computer processor, including laptop, handheld or mobile computers; vehicle-mounted devices; desktop computers; mainframe computers; etc., having processing (and optionally communication) capabilities. The remote server, similarly, can be one of any number of different types of computer which have processing and communication capabilities. These techniques are well known in the art, and the hardware devices and software which enable their use are readily available. The user's computer will be referred to equivalently as a "workstation", "device", or "computer", and use of any of these terms or the term "server" refers to any of the types of computing devices described above.

In the preferred embodiment, the present invention is implemented as one or more computer software programs. The implementation of the software for schema - driven compressed of XML or DOM files may operate on a server in a network, as one or more modules (also referred to as code subroutines, or "objects" in object-oriented programming) which are invoked upon request. Alternatively, the software may operate on a user's workstation. The logic implementing the invention may be integrated with the code of a program which creates the XML files or it may be implemented as one or more separate utility modules, without deviating from the inventive concepts disclosed herein. The server may be functioning as a Web server, where that Web server provides services in response to requests from a client connected through

the Internet. Alternatively, the server may be in a corporate intranet or extranet of which the client's workstation is a component, or in any other network environment. While the preferred embodiment anticipates that the compressed files are sent over a network connection, the file content may also be transferred between computers via a storage media (such as diskette), without deviating from the inventive concepts disclosed herein.

In addition to the hardware/software environment described above, a different aspect of the invention includes a computer-implemented method for performing the above method.

Such a method may be implemented, for example, by operating a computer, as embodied by a digital data processing apparatus, to execute a sequence of machine-readable instructions. These instructions may reside in various types of signal-bearing media.

The signal-bearing media may include, for example, a RAM contained within the CPU/processor, as represented by the fast-access storage for example. Alternatively, the instructions may be contained in another signal-bearing media, such as a magnetic data storage diskette 900 (Figure 9), directly or indirectly accessible by the CPU.

Whether contained in the diskette 900, the computer/CPU, or elsewhere, the instructions may be stored on a variety of machine-readable data storage media, such as DASD storage (e.g., a conventional "hard drive" or a RAID array), magnetic tape, electronic read-only memory (e.g., ROM, EPROM, or EEPROM), an optical storage device (e.g. CD-ROM, WORM, DVD, digital optical tape, etc.), paper "punch" cards, or other suitable signal-bearing media including transmission media such as digital and analog and communication links and wireless. In an illustrative embodiment of the invention, the machine-readable instructions may comprise software object code, compiled from a language such as "C", etc.

With the unique and unobvious features of the present invention, hence, the present invention not only takes advantage of the separation of the structure and data of an XML document, but it also takes advantage of the associated DTD (schema) of the document to perform optimization. The DTD schema describes the constraints on the structures, possible values, and occurrence restrictions of attribute values and elements. The compression algorithm, once it knows that there is a schema associated with the document, takes advantage of this and produces further compression of the data.

Further, the present invention provides an efficient compression algorithm for XML documents in which the XML documents are compressed, and such that the structural information will be kept in the compressed form so that the documents can be easily reconstructed. The invention provides a lossless compression algorithm that gets as close as possible to the ZLIB algorithm. Hence, with the invention, the markup (structure) and non-markup (data) can be separated , and the non-markup component can be compressed using ZLIB and the markup component can be compressed using binary encoding.

Thus, with the unique and unobvious aspects of the present invention, the schema information (the DTD associated with the document) can be used to compress the structure component and obtain higher compression rate while simultaneously retaining the structure.

While the invention has been described in terms of a preferred embodiment, those skilled in the art will recognize that the invention can be practiced with modification within the spirit and scope of the appended claims.

CLAIMS

Having thus described our invention, what we claim as new and desire to secure by Letters Patent is as follows:

1. A method of compressing an extensible markup language (XML) document, comprising:

compressing an XML document and its associated schema information such that information in a markup portion therein is maintained in a compressed form to allow the document to be reconstructed,

wherein, during said compressing, said markup portion and a non-markup portion of said document are separated, and the non-markup component is compressed using a first compression method and the markup component is compressed using a second compression method.

2. The method according to claim 1, wherein said mark-up portion comprises structured component information and wherein said schema information associated with the document is used with compressing the structure component to obtain a predetermined higher compression rate while simultaneously retaining the structure.

3. The method according to claim 2, wherein said schema information comprises a document type definition (DTD).

4. The method according to claim 1, wherein said markup portion comprises a structure of the document and said non-markup portion comprises data associated with said document.

5. The method according to claim 1, wherein said XML document includes elements selectively containing any of character data, child elements, or a combination of character data and child elements.

6. The method according to claim 1, wherein an XML markup language is defined in a Document Type Definition (DTD),

wherein said DTD is contained in at least one of a <!DOCTYPE> tag and an external file and referenced from a <!DOCTYPE> tag.

7. The method according to claim 1, wherein an element is defined as a group of one or more subelements, character data, EMPTY, or ANY, and

wherein attributes are optional, required, or selectively have a fixed value, and wherein optional attributes have a default and fixed attributes always have a default.

8. The method according to claim 6, wherein prior to compressing, a tag name token is associated with each tag name, an attribute name token is associated with each attribute name and an attribute value token is associated with each attribute value which is defined in the DTD.

9. The method according to claim 8, wherein a server searches for the token corresponding to an element or an attribute.

10. The method according to claim 9, wherein a client searches for the element or attribute corresponding to a particular token.

11. The method according to claim 10, wherein the server and the client each use a different data structure.

12. The method according to claim 1, wherein said compressing further comprises:

creating the data structure on a server and on a client.

13. The method according to claim 12, wherein said compressing further comprises:

parsing a document type definition (DTD) of the XML document.

14. The method according to claim 13, wherein said compressing further comprises:

filling in server code spaces.

15. The method according to claim 14, wherein said compressing further comprises:

filling in client code spaces.

16. The method according to claim 14, wherein, for a specific tag name or attribute name or value, the server finds a corresponding token.

17. The method according to claim 16, wherein a hash table is used for storing a key and a value as a couple where the key is a string and the value is a corresponding token.

18. The method according to claim 17, wherein the client finds the tag name or attribute name or value corresponding to a specific token, and wherein said token is split into a set of overlapping code spaces.

19. The method according to claim 18, wherein said client is provided with a page number and an index in a code space page, and finds the corresponding string, and wherein each code space includes a two-dimensional array indexed by page numbers and indexes in pages.

20. The method according to claim 14, wherein the DTD is parsed using a document object model (DOM) parser which generates a DOM structure for the DTD, for allowing accessing elements declarations and attributes declarations with name and type and attributes values when an attribute type is enumerated.

21. The method according to claim 14, wherein server code spaces are filled-in by filling in a hash table for an element code space such that a page number variable is set to a first predetermined value and the index variable is set to a second predetermined value.

22. The method according to claim 21, wherein, for each element declaration, an element name is added to the hash table with the element name as the key and (256 x pageNumber + index) as the value,

wherein the index is incremented by 1 and a size of a page for elements is a predetermined number and when the index reaches the predetermined number, the page number is incremented by 1 and the index is reset, and

wherein when the page number reaches a maximum value, an exception is raised.

23. The method according to claim 22, wherein for the attribute code space, for each element declared, a corresponding attribute declaration is obtained from a previously built Document Object Model (DOM) tree, and adds the attribute name in the hash table with the attribute name as the key and (256 x pageNumber + index) as the value.

24. The method according to claim 23, wherein, if the attribute type is enumerated, then the values of the enumerated attribute are searched,

for each value, the attribute value is added in the hash table with the attribute value as the key and (256 x pageNumber + index) as the value, and the index for the value is incremented,

wherein a size of a page for an attribute value is a predetermined number such that when the index reaches the predetermined number, the system increments the page number and resets the index, and

wherein, when the page number reaches its maximum value, an exception is raised,

wherein, if there are no values or when the values have been successfully added to the attribute value code space, then the index is incremented for the name,

wherein the size of a page for an attribute name is a predetermined number and such that when the index reaches the page number is incremented and the index is reset, and when the page number reaches its maximum value, an exception is raised.

25. The method according to claim 22, wherein the filling-in of the client code spaces includes:

setting the page number variable to a first predetermined number and the index variable to a second predetermined number,

for each element declaration, obtaining the element name, adding it in the elements array at a predetermined position indicating (page number, index) and incrementing the index.

26. The method according to claim 25, wherein for an attribute code space, for each element declared, obtaining the corresponding attribute declaration from a previously built DOM structure, adding the attribute name in the attribute names array at a predetermined position represented by a couple formed of (page number, index).

27. The method according to claim 26, wherein if the attribute type is enumerated, then checking for the values of this enumerated attribute, and

for each value, adding the attribute value in the attribute values array at a predetermined position (page number, index) and incrementing the index for the value,

wherein when the index reaches a predetermined value, incrementing the page number and resetting the index to a predetermined number, and

when the page number reaches a maximum value, raising an exception.

28. The method according to claim 27, wherein if there are no values or when the values have been added to the attribute value code space, incrementing the index for the name, and

wherein when the size of a page for attribute name is a predetermined number, incrementing the page number to a predetermined number and resetting the index to a predetermined number, and

when the page number reaches a maximum value, raising an exception.

29. The method according to claim 1, further comprising:

merging an attribute names code space and an attribute values code space into one token.

30. The method according to claim 1, wherein each couple formed by (attribute name, attribute value) is made into a single token.

31. The method according to claim 30, wherein, for each element declared, a corresponding attribute declaration is obtained, and wherein if an attribute-type is not enumerated by having a specific value declared for this attribute, then adding the attribute name in the attribute code space.

32. The method according to claim 31, wherein the attribute code space comprises a hash table for a server, and an array for a client.

33. The method according to claim 31, wherein if the attribute-type is enumerated, then values are searched for the enumerated attribute, and for each value, the couple (attribute name, attribute value) is added with a specific token in the attribute code space.

34. The method according to claim 33, wherein, when the server encounters an attribute with a value, the server searches the attribute code space for the couple (attribute name, attribute value), and

if the server finds the couple, the server sends the token associated therewith.

35. The method according to claim 34, wherein if the server cannot find the couple, the server looks for the attribute name in the attribute code space, and

wherein, if the name is found, the server sends the corresponding token for the name followed by a string inline token followed by the attribute value encoded in a charset specified at a beginning of the XML document stream.

36. The method according to claim 21, wherein the element code space includes elements selectively having required or fixed attributes, and wherein, for said elements having the required or fixed attributes tokens are not transmitted,

wherein the names of the required or fixed attributes with the element name are stored in the element code space.

37. The method according to claim 36, wherein to fill in the element code space, for each element declaration, the element name and required and fixed attributes are obtained, and

wherein the element names and the required and fixed attribute names are added to the element code space, and for the fixed attributes, a value thereof is added additionally.

38. The method according to claim 37, wherein, in the attribute code space, only implied attributes are stored with their value if defined.

39. The method according to claim 1, wherein said first compression method comprises a lossless data compression method.

40. The method according to claim 39, wherein said second compression method comprises a binary encoding method.

41. A method of compressing an extensible mark-up language (XML) document, comprising:

creating a data structure on a server and on a client;

parsing a document type definition (DTD) of the XML document;

filling in the server code spaces; and

filling in the client code spaces.

42. A system for compressing an extensible markup language (XML) document, comprising:

means for compressing an XML document such that information comprising a markup portion therein is maintained in a compressed form to allow the document to be reconstructed; and

means, during compressing, for separating said markup portion and a non-markup portion of said document, and wherein said compressing means compresses the non-markup component using a first compression method and compresses the markup component using a second compression method.

43. A programmable storage medium tangibly embodying a program of machine-readable instructions executable by a digital processing apparatus to perform compressing of an extensible markup language (XML) document, said method comprising:

compressing an XML document such that information comprising a markup portion therein is maintained in a compressed form to allow the document to be reconstructed,

wherein, during said compressing, said markup portion and a non-markup portion of said document are separated, and the non-markup component is compressed using a first compression method and the markup component is compressed using a second compression method.

44. A programmable storage medium tangibly embodying a program of machine-readable instructions executable by a digital processing apparatus to perform compressing of an extensible markup language (XML) document, said method comprising:

creating a data structure on a server and on a client;

parsing a document type definition (DTD) of the XML document;

filling in the server code spaces; and

filling in the client code spaces.

5

Sep 15 2002 09:54

SYSTEM AND METHOD FOR SCHEMA-DRIVEN COMPRESSION OF EXTENSIBLE MARK-UP LANGUAGE (XML) DOCUMENTS

ABSTRACT OF THE DISCLOSURE

A method (and system) for compressing an extensible markup language (XML) document, includes compressing an XML document such that information in a markup portion therein is maintained in a compressed form to allow the document to be reconstructed. During the compressing, the markup portion and a non-markup portion of the document are separated, and the non-markup component is compressed using a first compression method and the markup component is compressed using a second compression method.

Docket No. AM9-99-0161

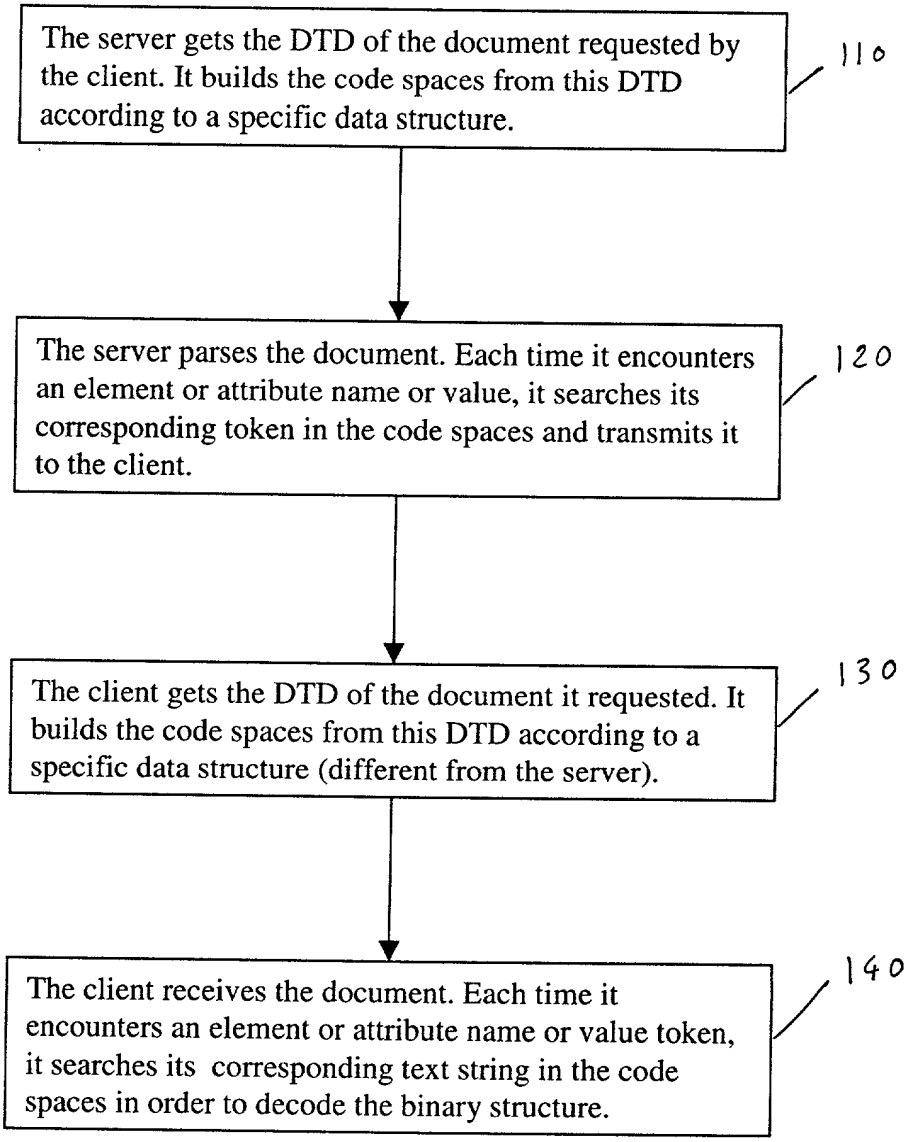

The server gets the DTD of the document requested by the client. It builds the code spaces from this DTD according to a specific data structure.
110
The server parses the document. Each time it encounters an element or attribute name or value, it searches its corresponding token in the code spaces and transmits it to the client.
120
The client gets the DTD of the document it requested. It builds the code spaces from this DTD according to a specific data structure (different from the server).
130
The client receives the document. Each time it encounters an element or attribute name or value token, it searches its corresponding text string in the code spaces in order to decode the binary structure.
140

Figure 1

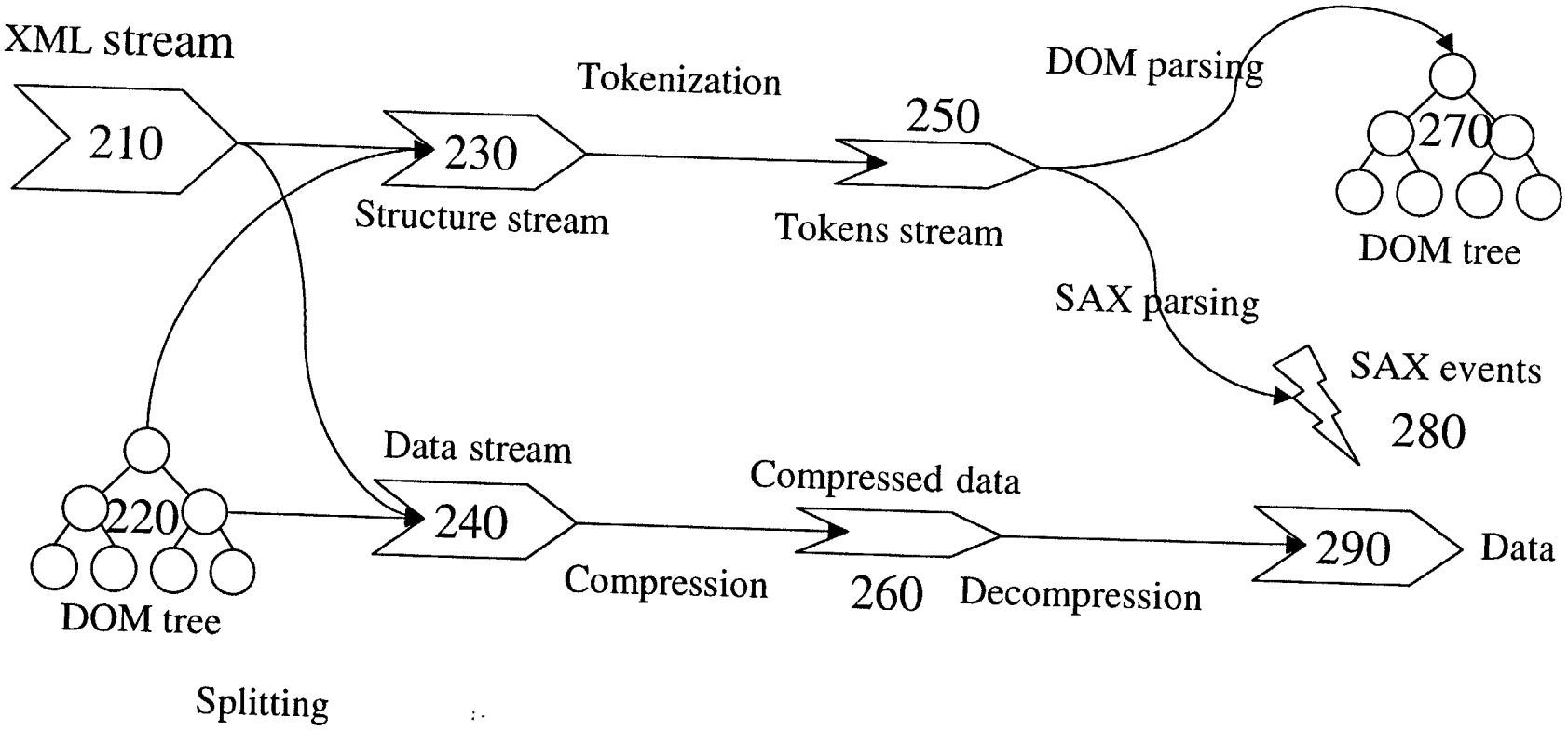

XML stream
210
Tokenization
230
Structure stream
250
Tokens stream
DOM parsing
270
DOM tree
SAX parsing
SAX events
280
Data stream
240
Compressed data
260
Compression
Decompression
290
Data
220
DOM tree
Splitting

Figure 2

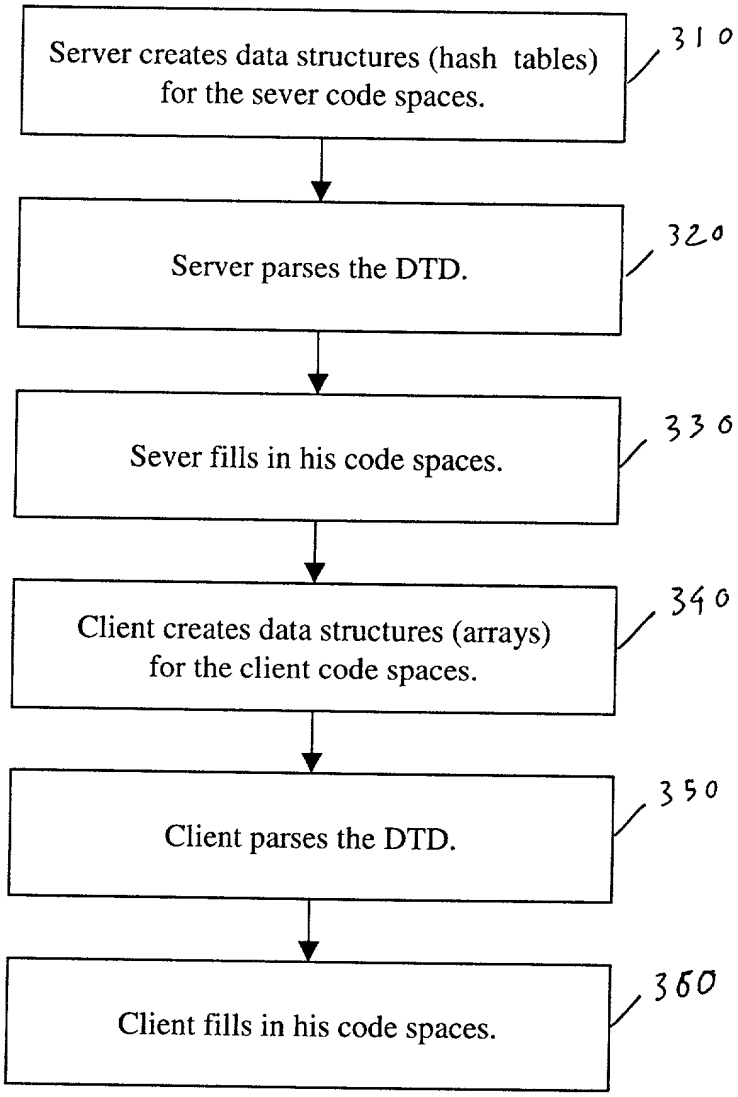

Server creates data structures (hash tables) for the sever code spaces.
310
Server parses the DTD.
320
Sever fills in his code spaces.
330
Client creates data structures (arrays) for the client code spaces.
340
Client parses the DTD.
350
Client fills in his code spaces.
360

Figure 3A

Tag code space		Attribute name code space		Attribute value code space	
Tag Name	Token	Attribute Name	Token	Attribute Value	Token
Book	5	Author	5	Literature	5
Title	6	Genre	6	Science	6
Chapter	7	Number	7	History	7
Picture	8	Caption	8	Cartoons	8

Hash table for the code spaces on the server side

Fig. 3B

index \ page	0	1	2	3	4
5	abbrev	codel	doctitle	glentry	index
6	abstract	codeno	dt	glossary	iref
7	address	codetext	dthd	group	kwd
8	annot	compl	epsc	grplbl	l
9	apl	coprext	erev	grpsep	label
10	appendix	coprnote	f	gt	lblbody
11	artalt	cover	fig	h0	lblbox
12	artdef	ct	figbody	h1	ldesc
13	artwork	danger	figcap	h2	le
14	asmlist	dataobj	figdef	h3	ledesc
15	author	date	figdesc	h4	ledi
16	autolink	dd	figlist	h5	legend
17	backm	ddhd	figref	h6	len
18	bibliog	delim	figseg	hd	lename
19	bin	dest	filenum	hdref	lers
20	bindnum	df	fle	hex	lersdef
21	body	dfdef	fn	hp0	li
22	c	dialdef	fnref	hp1	library
23	cause	dialog	fragment	hp2	lines
24	caution	dirdef	fragref	hp3	link
25	cgdef	divbody	frontm	hp4	liref
26	cgraphic	dl	ft	i1	lp
27	char	dldef	gd	i2	lq
28	ci	dlentry	gdg	i3	map
29	cit	docdesc	gendtitle	ih1	mapqfix
30	cmt	docnum	gl	ih2	mcdef
31	code	docprof	gldef	ih3	mcdi

Fig. 3C

Code Space Array

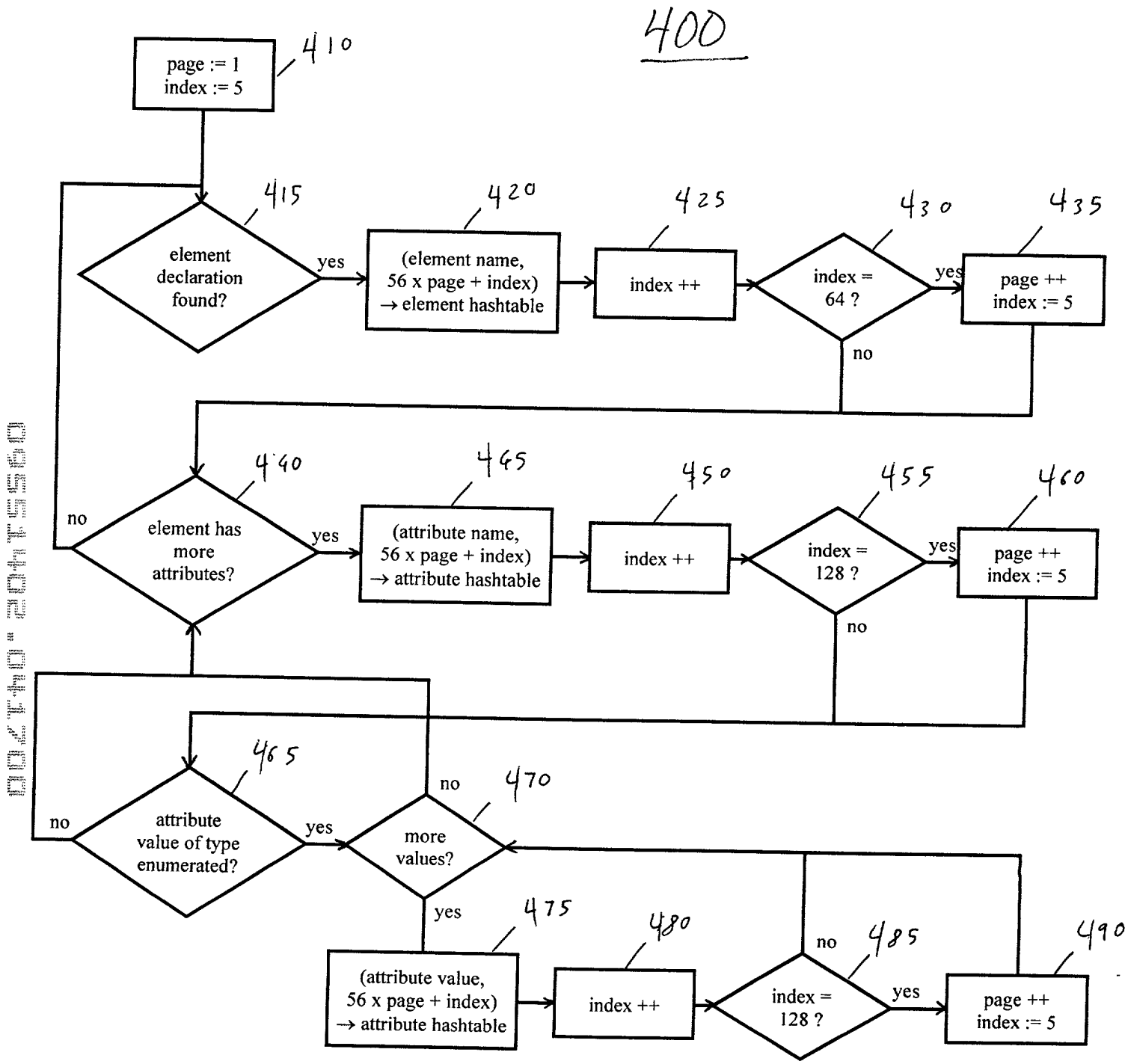

400
page := 1
index := 5
410
element declaration found?
415
yes
(element name, 56 x page + index) → element hashtable
420
index ++
425
index = 64 ?
430
yes
page ++ index := 5
435
no
element has more attributes?
440
no
yes
(attribute name, 56 x page + index) → attribute hashtable
445
index ++
450
index = 128 ?
455
yes
page ++ index := 5
460
no
attribute value of type enumerated?
465
no
yes
more values?
470
no
yes
(attribute value, 56 x page + index) → attribute hashtable
475
index ++
480
index = 128 ?
485
no
yes
page ++ index := 5
490

Figure 4

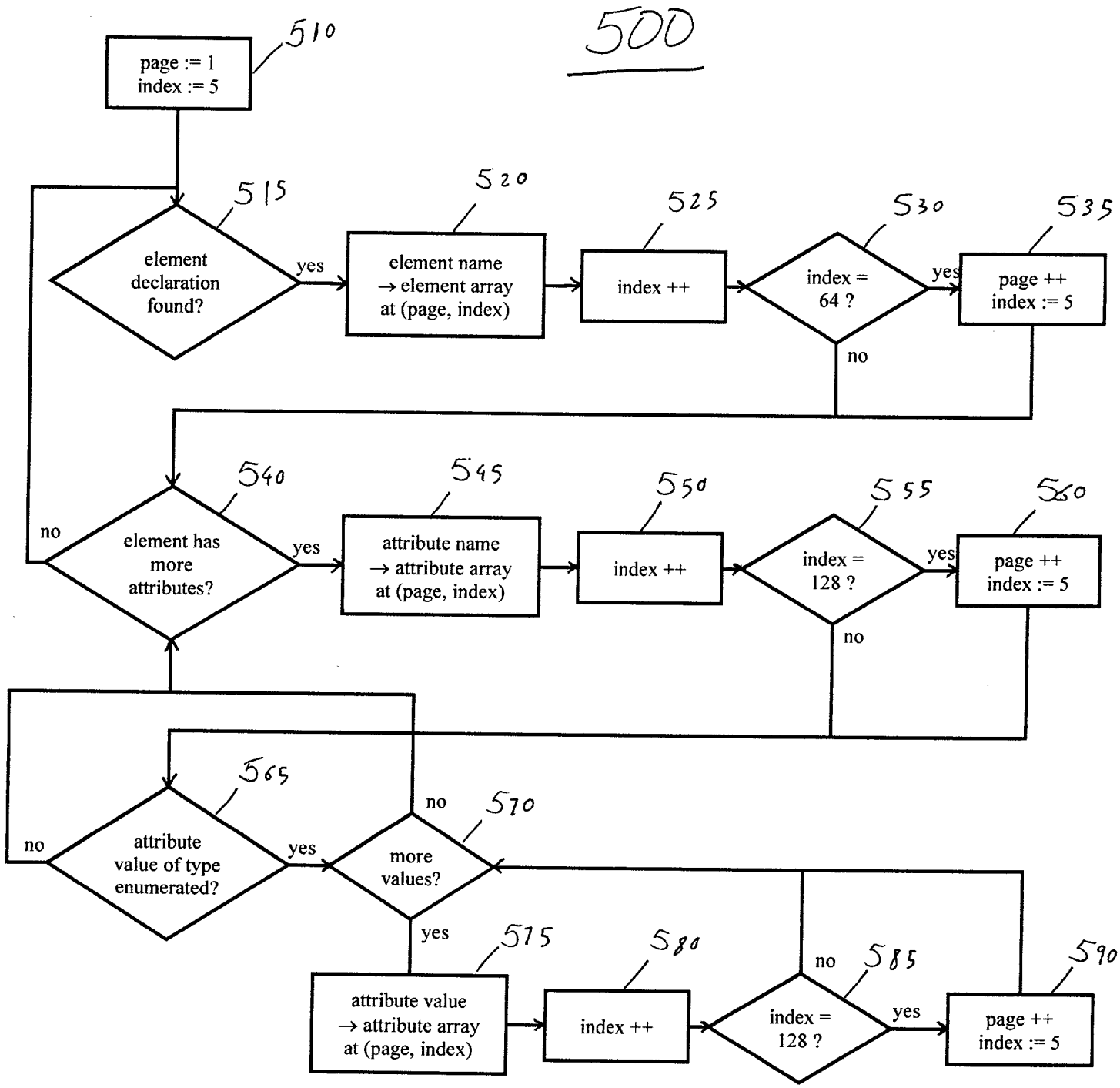

500
510
page := 1
index := 5
515
element declaration found?
yes
520
element name → element array at (page, index)
525
index ++
530
index = 64 ?
yes
535
page ++
index := 5
no
540
element has more attributes?
no
yes
545
attribute name → attribute array at (page, index)
550
index ++
555
index = 128 ?
yes
560
page ++
index := 5
no
565
attribute value of type enumerated?
no
yes
570
more values?
no
yes
575
attribute value → attribute array at (page, index)
580
index ++
585
index = 128 ?
no
yes
590
page ++
index := 5

Figure 5

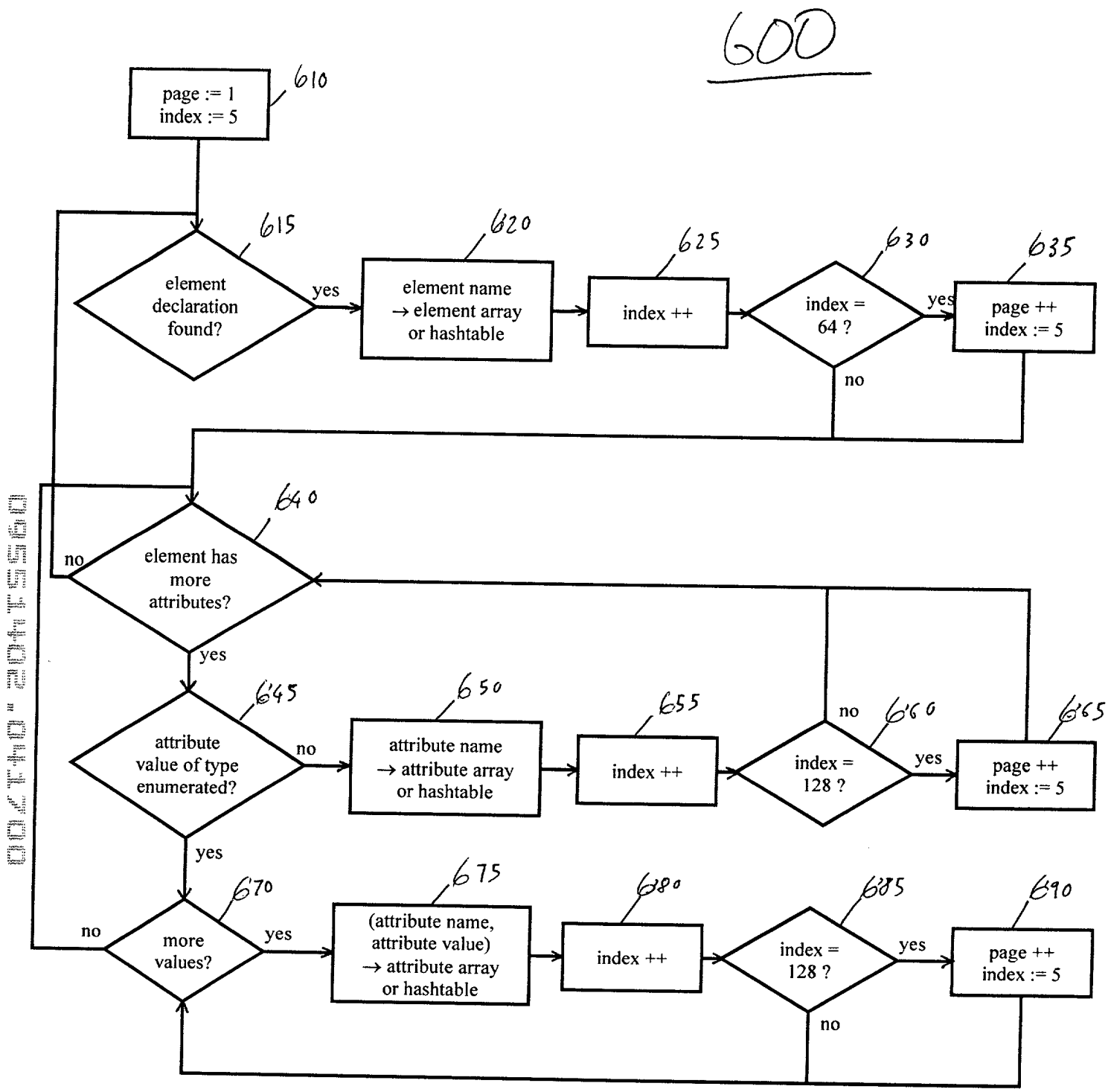

600
page := 1
index := 5
610
element declaration found?
615
yes
element name → element array or hashtable
620
index ++
625
index = 64 ?
630
yes
page ++ index := 5
635
no
element has more attributes?
640
no
yes
attribute value of type enumerated?
645
no
attribute name → attribute array or hashtable
650
index ++
655
index = 128 ?
660
no
yes
page ++ index := 5
665
yes
more values?
670
no
yes
(attribute name, attribute value) → attribute array or hashtable
675
index ++
680
index = 128 ?
685
yes
page ++ index := 5
690
no

Figure 6

0027h0" 20hT5560

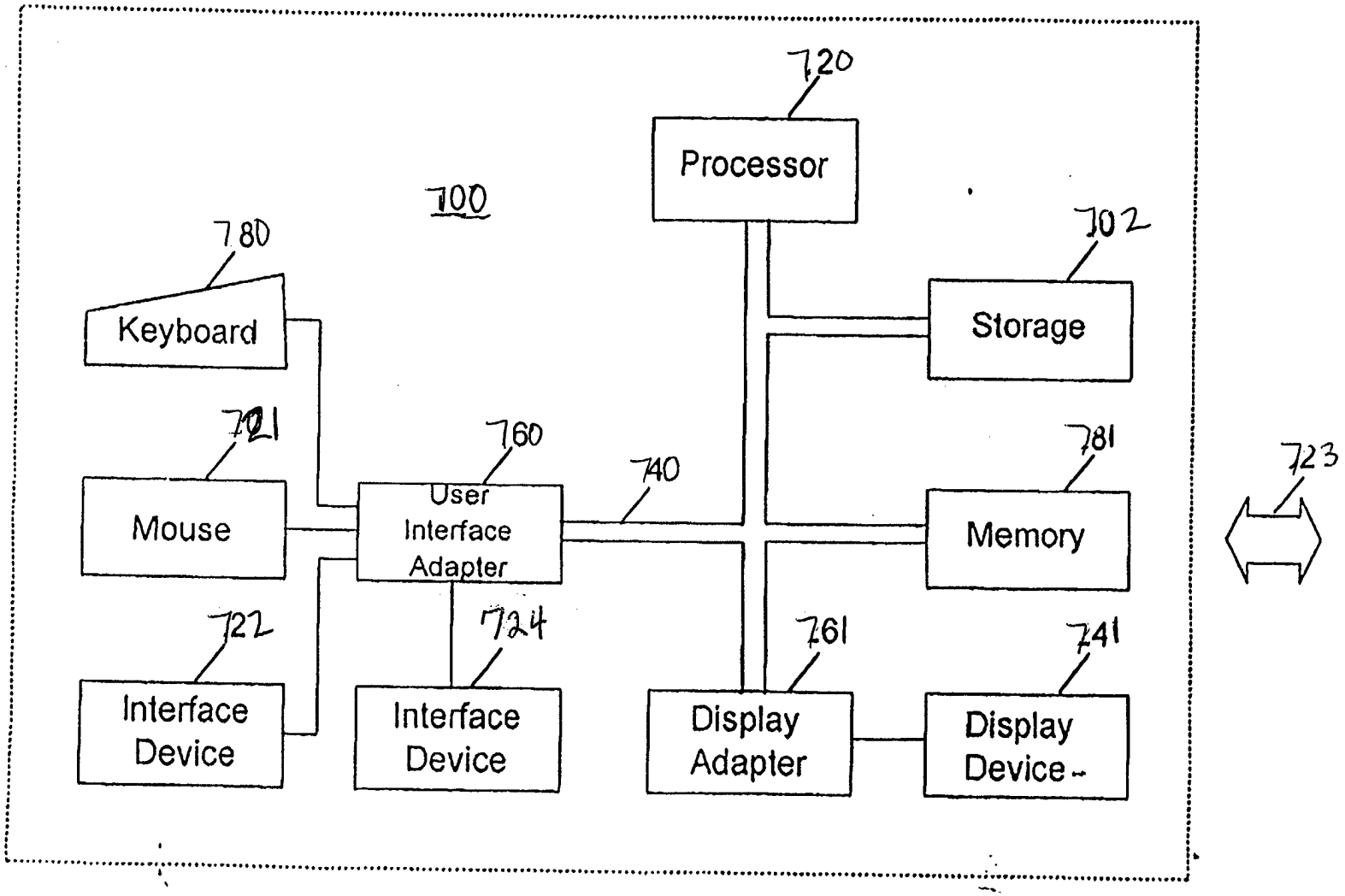


Fig. 7

0827h0" 20hT9560

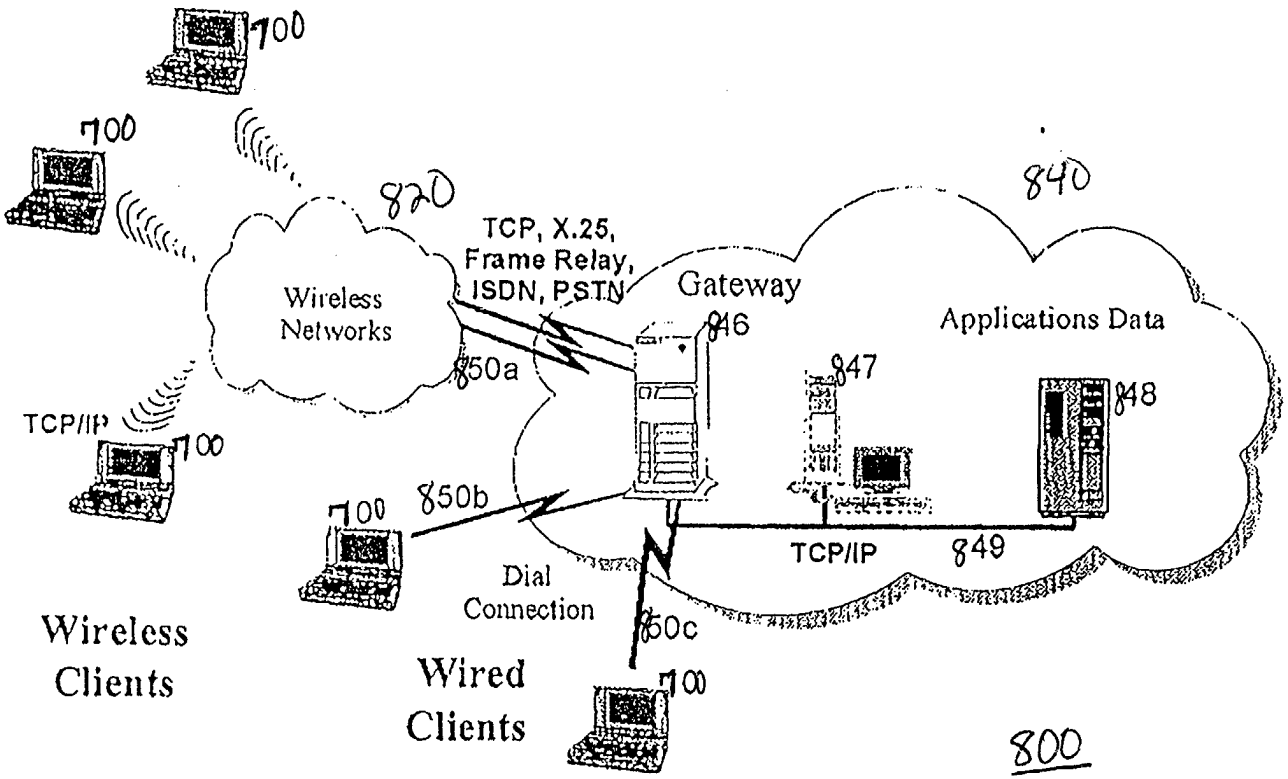
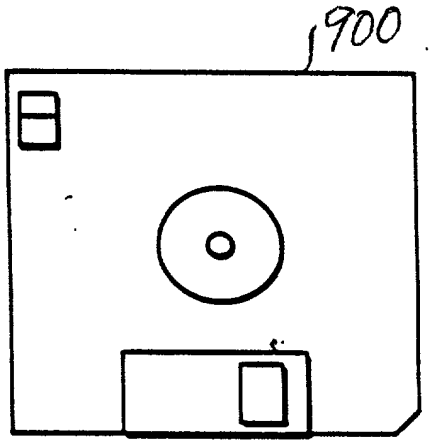


Fig. 8

FIG 9



SUBSTITUTE SHEET (RULE 26)

IBM Docket No.: AM9-99-0161

DECLARATION AND POWER OF ATTORNEY

As a below named inventor, I hereby declare that:

My residence, post office address and citizenship are as stated below next to my name; I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled: SYSTEM AND METHOD FOR SCHEMA-DRIVEN COMPRESSION OF EXTENSIBLE MARK-UP LANGUAGE (XML) DOCUMENTS

the specification of which:
(check one)

- ☒ is attached hereto.
- ☐ was filed on _____, as Application Serial No. _____ and was amended on _____.

I hereby state that I have reviewed and understand the contents of the above identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose information which is material to the patentability of this application in accordance with Title 37, Code of Federal Regulations, § 1.56.

I hereby claim foreign priority benefits under Title 35, United States Code, § 119 of any foreign application(s) for patent or inventor's certificate listed below and have also identified below any foreign application for patent or inventor's certificate having a filing date before that of the application on which priority is claimed:

Prior Foreign Application(s):

Number	Country	Day/Month/Year	Priority Claimed

I hereby claim the benefit under Title 35, United States Code, § 120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code, § 112, I acknowledge the duty to disclose material information as defined in Title 37, Code of Federal Regulations, § 1.56 which occurred between the filing date of the prior application and the national or PCT international filing date of this application:

Prior U.S. Applications:

Serial No.	Filing Date	Status

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

As a named inventor, I hereby appoint the following attorneys and/or agents to prosecute this application and transact all business in the Patent and Trademark Office connected therewith: We hereby appoint Khanh Q. Tran, Registration No. 41,352, Thomas R. Berthold, Registration No. 28,689, Marc McSwain, Registration No. 44,929, Alison D. Mortinger, Registration No. 39,306, Richard Ludwin, Registration No. 33,010, Sean M. McGinn, Registration No. 34,386, and Frederick W. Gibb, III, Registration No. 37,629 to prosecute this application and transact all business in the United States Patent and Trademark Office connected therewith.

Send all correspondence to: **McGinn & Gibb, P.C., 1701 Clarendon Boulevard, Suite 100, Arlington, Virginia 22209. Customer No. 21254**

Telephone calls should be directed to Sean M. McGinn, McGinn & Gibb, P.C. at (703) 294-6699.

(1) Inventor: Marc Georges Girardot

Signature: Marc Girardot Date: 03/20/2000

Residence: 18 rue Gay-Lussac, Paris, France 75005

Citizenship: France

Post Office Address: Same as Residence

Docket #AM9-99-0161

(2) Inventor: Neelakantan Sundaresan

Signature: _____ Date: _____

Residence: 492 Capital Village Circle, San Jose, CA 95136

Citizenship: India

Post Office Address: Same as Residence

Docket No. AM9-99-0161

IBM Docket No.: AM9-99-0161

DECLARATION AND POWER OF ATTORNEY

As a below named inventor, I hereby declare that:

My residence, post office address and citizenship are as stated below next to my name; I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled: SYSTEM AND METHOD FOR SCHEMA-DRIVEN COMPRESSION OF EXTENSIBLE MARK-UP LANGUAGE (XML) DOCUMENTS

the specification of which:
(check one)

☒ is attached hereto.
☐ was filed on _____, as Application Serial No. _____ and was amended on _____.

I hereby state that I have reviewed and understand the contents of the above identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose information which is material to the patentability of this application in accordance with Title 37, Code of Federal Regulations, § 1.56.

I hereby claim foreign priority benefits under Title 35, United States Code, § 119 of any foreign application(s) for patent or inventor's certificate listed below and have also identified below any foreign application for patent or inventor's certificate having a filing date before that of the application on which priority is claimed:

Prior Foreign Application(s):

Number	Country	Day/Month/Year	Priority Claimed

I hereby claim the benefit under Title 35, United States Code, § 120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code, § 112, I acknowledge the duty to disclose material information as defined in Title 37, Code of Federal Regulations, § 1.56 which occurred between the filing date of the prior application and the national or PCT international filing date of this application:

Prior U.S. Applications:

Serial No.	Filing Date	Status

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

As a named inventor, I hereby appoint the following attorneys and/or agents to prosecute this application and transact all business in the Patent and Trademark Office connected therewith: We hereby appoint Khanh Q. Tran, Registration No. 41,352, Thomas R. Berthold, Registration No. 28,689, Marc McSwain, Registration No. 44,929, Alison D. Mortinger, Registration No. 39,306, Richard Ludwin, Registration No. 33,010, Sean M. McGinn, Registration No. 34,386, and Frederick W. Gibb, III, Registration No. 37,629 to prosecute this application and transact all business in the United States Patent and Trademark Office connected therewith.

Send all correspondence to: **McGinn & Gibb, P.C., 1701 Clarendon Boulevard, Suite 100, Arlington, Virginia 22209. Customer No. 21254**

Telephone calls should be directed to Sean M. McGinn, McGinn & Gibb, P.C. at (703) 294-6699.

(1) Inventor: Marc Georges Girardot

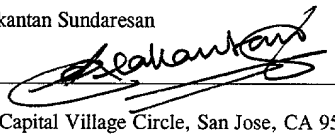
Signature: _____ Date: _____

Residence: 18 rue Gay-Lussac, Paris, France 75005

Citizenship: France

Post Office Address: Same as Residence

(2) Inventor: Neelakantan Sundaresan

Signature:  Date: 03/22/2000

Residence: 492 Capital Village Circle, San Jose, CA 95136

Citizenship: India

Post Office Address: Same as Residence

Sheet 2 of 2 - Oath